Introduction:
The dataset used for this project is the Home Loan Approval dataset. The Home Loan Approval dataset id sourced from Kaggle. It is a comprehensive collection of data tailored for evaluating factors influencing home loan approval. The dataset encompasses diverse features, including applicant demographics, financial metrics, property details, and loan-related information. Notably, it contains both numerical and categorical features, capturing a wide range of variables crucial in assessing loan eligibility. The target variable, "Loan Approval," indicates whether a home loan application was approved or not.
The dataset comprises 13 rows and 615 columns. The data includes details such as applicant income, credit history, loan amount, property location, and other pertinent factors. This extensive dataset provides a rich source of information for in-depth analysis and modeling in the context of home loan approval.

Task 1:
Task 1 involves reading and loading the dataset file into the program using the Pandas library. Additionally, the dataset is shuffled using the shuffle function from the numpy library.

```
np.random.seed(123)
df = pd.read_csv('/content/drive/My
Drive/python_final_project_group_05/home_loan_approval.csv')
df = shuffle(df)
print(df)
```

The code ensures reproducibility by setting a random seed, reads the dataset from the specified file path using Pandas, shuffles the rows.

Task 2:
Task 2 involves applying appropriate data cleaning techniques to the home loan approval dataset. The Pandas library is employed to replace missing values using suitable methods. Duplicate records are identified and removed.

```
x=df['Gender'].mode()[0]
df['Gender'].fillna(x,inplace=True)

x=df['Married'].mode()[0]
df['Married'].fillna(x,inplace=True)

x=df['Dependents'].mode()[0]
df['Dependents'].fillna(x,inplace=True)

x=df['Self_Employed'].mode()[0]
df['Self_Employed'].fillna(x,inplace=True)

x=df['LoanAmount'].mean()
df['LoanAmount'].fillna(x,inplace=True)
```

```
x=df['Loan_Amount_Term'].mode()[0]
df['Loan_Amount_Term'].fillna(x,inplace=True)

x=df['Credit_History'].mode()[0]
df['Credit_History'].fillna(x,inplace=True)

duplicate = df.duplicated().any()
print("Duplicate Rows: \n", duplicate)

bins = [0, 30000, 60000, 90000, float('inf')]
labels = ['0-30k', '30k-60k', '60k-90k','90k+']
```

The code replaces missing values in relevant columns using statistical measures like mode and mean. Duplicate rows are checked and removed, ensuring data integrity. The bins and labels for categorizing 'Income' are also defined in this step.

Task 3: Task 3 involves visualizing the frequency distributions of various features in the home loan approval dataset using the Matplotlib library.

```
plt.figure(figsize=(22, 20))

i = 1
for column in df.columns[1:]:
  frequency = df[column].value_counts()

  plt.subplot(4, 3, i)
  plt.bar(frequency.index, frequency.values)
  plt.xlabel(column)
  i = i+1

plt.show()
```

In this code, a figure of size (22, 20) is created, and a loop iterates through each feature in the dataset, generating subplots for each feature's frequency distribution. Bar plots are used to visualize the count of each unique value in the respective feature. The plt.show() function is then called to display all the plots in a single diagram.

Task 4:
Task 4 involves visualizing relationships between the target column ('Loan_Status') and other columns in the home loan approval dataset using the Matplotlib library. The subplot() function is utilized to display all plots in a single figure.

```
plt.figure(figsize=(20, 12))
target_column = 'Loan_Status'

i = 1
for column in df.columns[1:12]:
  plt.subplot(4, 3, i)
  plt.scatter(df[column], df[target_column])
  plt.title(f'Loan_Status and {column}')
  i = i+1

plt.show()
```

In this code, a figure of size (20, 12) is created, and a loop iterates through each column in the dataset. Scatter plots are used to visualize the relationship between each feature and the target column is Loan_Status. The plt.show() function is then called to display all the plots in a single diagram.

Task 5:
Task 5 involves performing feature scaling on the dataset features. The LabelEncoder is applied to convert categorical variables, and then StandardScaler is used to scale the numerical features.

```
label_encoder = LabelEncoder()
df['Gender'] = label_encoder.fit_transform(df['Gender'])
df['Married'] = label_encoder.fit_transform(df['Married'])
df['Dependents'] = label_encoder.fit_transform(df['Dependents'])
df['Education'] = label_encoder.fit_transform(df['Education'])
df['Property_Area'] = label_encoder.fit_transform(df['Property_Area'])
df['Self_Employed'] = label_encoder.fit_transform(df['Self_Employed'])


features = df.columns[1:12]

X = df[features]
y = df['Loan_Status']

scaler = StandardScaler()
X[features] = scaler.fit_transform(X)
print(X)
```

In this code, categorical variables are converted using LabelEncoder, and then StandardScaler is applied to scale the numerical features. The resulting scaled features are stored in the variable X.

Task 6:
Task 6 involves splitting the dataset into training and testing sets using the train_test_split() function. The random_state parameter is set to 123 for reproducibility.

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=123)

print("Training:", X_train.shape, y_train.shape)
print("Testing:", X_test.shape, y_test.shape)
```

In this code, the train_test_split() function is utilized to split the features (X) and the target variable (y) into training and testing sets. The test_size parameter is set to 0.2, indicating an 80-20 split. The random_state parameter is set to 123 for reproducibility. The shapes of the resulting training and testing sets are then printed for verification.

Task 7:
Task 7 involves applying the Naïve Bayes Classifier to the dataset. The Gaussian Naive Bayes model is used, and the model is trained on the training set. Predictions are then made on the testing set.

```
nb_model = GaussianNB()

nb_model.fit(X_train, y_train)

y_prediction = nb_model.predict(X_test)
```

In this code, a Gaussian Naive Bayes model is created using GaussianNB(). The fit() method is then used to train the model on the training set (X_train, y_train). Finally, predictions are made on the testing set (X_test), and the predicted values are stored in the variable y_prediction.

Task 8:
Task 8 involves calculating the confusion matrix for the Naïve Bayes Classifier model and interpreting it in detail in the report. The metrics.confusion_matrix function is used, and the results are visualized using ConfusionMatrixDisplay.

```
confusion_matrix = metrics.confusion_matrix(y_test, y_prediction)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix =
confusion_matrix,
                                            display_labels =
nb_model.classes_)

cm_display.plot()
plt.title('Confusion Matrix')
plt.show()
```

In this code, the confusion matrix is calculated using metrics.confusion_matrix by comparing the actual values (y_test) with the predicted values (y_prediction). The ConfusionMatrixDisplay is then used to visualize the confusion matrix, and the plot is displayed with a title indicating 'Confusion Matrix.'

Task 9:
Task 9 involves calculating the accuracy, precision, recall, and F1 score of the Naïve Bayes Classifier model. The accuracy_score, precision_score, recall_score, and f1_score functions from the sklearn.metrics module are used.

```
pos_label = 'Y'
```

```
accuracy = accuracy_score(y_test, y_prediction)
print("Accuracy:", accuracy)

precision = precision_score(y_test, y_prediction, pos_label=pos_label)
print("Precision:", precision)

recall = recall_score(y_test, y_prediction, pos_label=pos_label)
print("Recall:", recall)

f1_score = f1_score(y_test, y_prediction, pos_label=pos_label)
print("F1 Score:", f1_score)
```

In this code, the accuracy, precision, recall, and F1 score are calculated using the appropriate metrics functions. The results are then printed to assess the overall performance of the Naïve Bayes Classifier model. These metrics provide insights into the model's ability to correctly classify instances and handle class imbalances.

Task 10:

Task 10 involves demonstrating how 10-fold cross-validation can be used to build a Naïve Bayes Classifier and reporting the accuracy of this model. The cross_val_score function from the sklearn.model_selection module is employed.

```
num_folds = 10

accuracy_scores = cross_val_score(nb_model, X, y, cv=num_folds,
scoring='accuracy')

mean_accuracy = np.mean(accuracy_scores)
print(f'Mean Accuracy across {num_folds}-fold cross-validation:
{mean_accuracy:.4f}')
```

In this code, the cross_val_score function is used to perform 10-fold cross-validation on the Naïve Bayes Classifier model (nb_model). The accuracy scores for each fold are calculated, and the mean accuracy across all folds is printed. This provides a robust assessment of the model's performance across different subsets of the data